A hybrid classical-quantum workflow for natural language processing

IOP Publishing Mach. Learn.: Sci. Technol. 2 (2021) 015011 https://doi.org/10.1088/2632-2153/abbd2e

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
7 May 2020

REVISED
1 September 2020

ACCEPTED FOR PUBLICATION
30 September 2020

PUBLISHED
1 December 2020

PAPER

A hybrid classical-quantum workflow for natural language processing

Lee J O'Riordan[1,2], **Myles Doyle**[1,2], **Fabio Baruffa**[3] **and Venkatesh Kannan**[1,2]

[1] Irish Centre for High-End Computing, Dublin, Ireland
[2] National University of Ireland, Galway, Ireland
[3] Intel Deutschland GmbH, Feldkirchen, Germany

E-mail: lee.oriordan@ichec.ie

Keywords: quantum computing, NLP, AI, HPC

Abstract

Natural language processing (NLP) problems are ubiquitous in classical computing, where they often require significant computational resources to infer sentence meanings. With the appearance of quantum computing hardware and simulators, it is worth developing methods to examine such problems on these platforms. In this manuscript we demonstrate the use of quantum computing models to perform NLP tasks, where we represent corpus meanings, and perform comparisons between sentences of a given structure. We develop a hybrid workflow for representing small and large scale corpus data sets to be encoded, processed, and decoded using a quantum circuit model. In addition, we provide our results showing the efficacy of the method, and release our developed toolkit as an open software suite.

1. Introduction

Natural language processing (NLP) is an active area of both theoretical and applied research, and covers a wide variety of topics from computer science, software engineering, and linguistics, amongst others. NLP is often used to perform tasks such as machine translation, sentiment analysis, relationship extraction, word sense disambiguation and automatic summary generation [1]. Most traditional NLP algorithms for these problems are defined to operate over strings of words, and are commonly referred to as the 'bag of words' approach [2]. The challenge, and thus limitation, of this approach is that the algorithms analyse sentences in a corpus based on meanings of the component words, and lack information from the grammatical rules and nuances of the language. Consequently, the qualities of results from these traditional algorithms are often unsatisfactory when the complexity of the problem increases.

On the other hand, an alternate approach called 'compositional semantics' incorporates the grammatical structure of sentences from a given language into the analysis algorithms. Compositional semantics algorithms include the information flows between words in a sentence to determine the meaning of the whole sentence [3]. One such model in this class is '(categorical) distributional compositional semantics', known as DisCoCat [4–6], which is based on tensor product composition to give a grammatically informed algorithm that computes the meaning of sentences and phrases. This algorithm has been noted to potentially offer improvements to the quality of results, particularly for more complex sentences, in terms of memory and computational requirements. However, the main challenge in its implementation is the need for large classical computational resources.

With the advent of quantum computer programming environments, both simulated and physical, a question may be whether one can exploit the available Hilbert space of such systems to carry out NLP tasks. The DisCoCat methods have a natural extension to a quantum mechanical representation, allowing for a problem to be mapped directly to this formalism [5]. Using an oracle-based access pattern, one can bound the number of accesses required to create the appropriate states for use by the DisCoCat methods [7]. Though, this requires the use of a quantum random access memory, or qRAM [8, 9]. Currently, qRAM remains unrealised, and expectations are that the resources necessary to realise are as challenging as a fault

tolerant quantum computer [10]. As such, it can be useful to examine scenarios where qRAM is not part of the architectural design of the quantum circuit. This will allow us to examine proof-of-concept methods to explore and develop use-cases later improved by its existence.

In this paper we examine the process for mapping a corpus to a quantum circuit model, and use the encoded meaning-space of the corpus to represent fundamental sentence meanings. With this representation we can examine the mapping of sentences to the encoding space, and additionally compare sentences with overlapping meaning-spaces. We follow a DisCoCat-inspired formalism to define sentence meaning and similarity based upon a given compositional sentence structure, and relationships between sentence tokens determined using a distributional method of token adjacency.

This paper will be laid out as follows. Section 2 will give an introduction to NLP, the application of quantum models to NLP, and discuss the encoding strategy for a quantum circuit model. Section 3 will discuss the preparation methods required to enable quantum-assisted encoding and processing of the text data. Section 4 will demonstrate the proposed methods using our quantum NLP software toolkit [11] sitting atop Intel Quantum Simulator (IQS) [12]. For this we showcase the methods, and compare results for corpora of different sizes and complexity. Finally, we conclude in section 5.

2. NLP methods

One of the main concerns of NLP methods is the extraction of information from a body of text, wherein the data is not explicitly structured; generally, the text is meant for human, rather than machine, consumption [13]. As such, explicit methods to infer meaning and understand a body of text are required to encode such data in a computational model.

Word embedding models, such as `word2vec`, have grown in popularity due to their success in representing and comparing data using vectors of real numbers [14]. Additionally, libraries and toolkits such as NLTK [15] and spaCy [16] offer community developed models and generally incorporate the latest research methods for NLP. The use of quantum mechanical effects for embedding and retrieving information in NLP has seen much interest in recent years [17–24].

An approach that aims to overcome the ambiguity offered by traditional NLP methods, such as the bag-of-words model is the categorical distributional-compositional (DisCoCat) model [4, 5]. This method incorporates semantic structure, where sentences are constructed through a natural tensoring of individual component words following a set of rules determined from category theory. These rule-sets for which sentence structures may be composed are largely based on the framework of pre-group grammars [25].

The DisCoCat approach offers a means to employ grammatical structure of sentences with token relationships in these sentences. Words that appear closer in texts are more likely to be related, and sentence structures can be determined using pre-group methods. These methods can easily be represented in a diagrammatic form, and allow for a natural extension to quantum state representation [6]. This diagrammatic form, akin to a tensor network, allows for calculating the similarity between other sentences. This similarity measure assumes an encoded quantum state representing the structure of the given corpus, and an appropriately prepared test state to compare with. This alludes to a tensor-contraction approach to perform the evaluation.

While this approach has advantages in terms of accuracy and generalisation to complex sentence structures, state preparation is something we must consider. Given the current lack of qRAM, the specified access bounds are unrealised [7], and so it is worth considering state preparation as part of the process. Ensuring an efficient preparation approach will also be important to enable processing on a scale to rival that of traditional high-performance computing NLP methods.

As such, we aim to provide a simplified model, framework and hybrid workflow for representing textual data using a quantum circuit model. We draw inspiration from the DisCoCat model to preprocess our data to a structure easily implementable on a quantum computer. We consider simple sentences of the form 'noun—verb—noun' to demonstrate this approach. All quantum circuit simulations and preprocessing is performed by our quantum NLP toolkit (QNLP), sitting atop the Intel Quantum Simulator (formerly qHiPSTER) to handle the distributed high-performance quantum circuit workloads [12, 26]. We release our QNLP toolkit as an open source (Apache 2.0) project, and have made it available on GitHub [11].

3. Methods

3.1. Representing meaning in quantum states

In this section, we discuss the implementation of the algorithms required to enable encoding, processing, and decoding of our data. We consider a simplified restricted example of the sentence structure

'noun—verb—noun' as the representative encoding format. To represent sentence meanings using this workflow, we must first consider several steps to prepare our corpus data set for analysis:

(a) Data must be pre-processed to tag tokens with the appropriate grammatical type; stop-words (e.g. 'the', 'a', 'at', etc) and problematic (e.g. non-alphanumeric) characters should be cleaned from the text to ensure accurate tagging, wherein type information is associated with each word.
(b) The pre-processed data must be represented in an accessible/addressable (classical) memory medium.
(c) There must be a bijective mapping between the pre-processed data and the quantum circuit representation to allow both encoding and decoding.

Assuming an appropriately prepared dataset, the encoding of classical data into a quantum system can be mapped to two different approaches: state (digital), or amplitude (analogue) encoding [27, 28]. We aim to operate in a mixed-mode approach: encoding and representing corpus data using state methods, then representing and comparing test sentence data through amplitude adjustment, measurement, and overlap.

Our approach to encoding data starts with defining a fundamental language (basis) token set for each representative token meaning space (subject nouns, verbs, object nouns). The notion of similarity, and hence orthogonality, with language can be a difficult problem. Do we consider the words 'stand' and 'sit' to be completely opposite, or are they similar because of the type of action taken? For this work, we let the degree of 'closeness' be determined by the distributional nature of the terms in the corpus; words further apart in the corpus are more likely to be opposite.

To efficiently encode the corpus data, we decide to represent the corpus in terms of the n most fundamentally common tokens in each meaning space. This draws similarity with the use of a word embedding model to represent a larger space of tokens in terms of related meanings in a smaller space [29–31]. This is necessary as representing each token in the corpus matching the sentence structure type can create a much larger meaning space than is currently representable, given realistic simulation constraints. However, one can note as we increase the limit of fundamental tokens in our basis, we tend to the full representative meaning model.

Taking inspiration from the above methods, we implement an encoding strategy that given the basis tokens, maps the remaining non-basis tokens to these, given some distance cut-off in the corpus. A generalised representation of each token t_i, in their respective meaning space $\mathbf{m}$ would be defined as

$$t_i = \sum_{j}^{n} d_{i,j} m_j, \tag{1}$$

where $d_{i,j}$ defines the distance between the base token m_j and non-base t_i. As such, we obtain a linear combination of the base tokens with representative weights to describe the mapped tokens.

We have identified the following key steps to effectively pre-process data for encoding:

(a) Tokenise the corpus and record position of occurrence in the text.
(b) Tag tokens with the appropriate meaning space type (e.g. noun, verb, stop-word, etc)
(c) Separate tokens into noun and verb datasets.
(d) Define basis tokens in each set as the N_{nouns} and N_{verbs} most frequently occurring tokens.
(e) Map basis tokens in each respective space to a fully connected graph, with edge weights defined by the minimum distance between each other basis token.
(f) Calculate the shortest Hamiltonian cycle for the above graph. The token order within the cycle is reflective of the tokens' separation within the text, and a measure of their similarity.
(g) Map the basis tokens to binary strings, using a given encoding scheme.
(h) Project composite tokens (i.e. non-basis tokens) onto the basis tokens set using representation cut-off distances for similarity, W_{nouns} and W_{verbs}.
(i) Form sentences by matching composite NOUN-VERB-NOUN tokens using relative distances and a NOUN-VERB distance cut-off, W_{nv}.

After conducting the pre-processing steps, the corpus is represented as a series of binary strings of basis tokens. At this stage the corpus is considered prepared and can be encoded into a quantum register.

3.2. Token encoding

To ensure the mapping of basis words to encoding patterns is reflective of the underlying distributional relationships between words in the corpus, it is necessary to choose an encoding scheme such that the inter-token relationships are preserved. While many more complex schemes can give insightful relationships,

we choose a cyclical unary encoding scheme where the Hamming distance, d_H, between each bit-string is equal to the distance between the bit-strings in the data set. Given an n-qubit register, we generate P_n, a list of $2n$ unique bit-patterns represented as integers, as defined by algorithm 1.

Algorithm 1: Generating unique patterns for token-to-state mappings.

Input: Number of qubits, n
Output: List P, with $2n$ unique patterns

1: **function** PATTERNS(n)
2: $P \leftarrow [\,]$
3: **for** $i \leftarrow 0$ to $2n-1$ **do**
4: **if** $i \leq n$ **then**
5: $P[i] = 2^i - 1$
6: **else**
7: $P[i] = 2^n - 2^{i \mod n}$
8: **end if**
9: **end for**
10: **return** P
11: **end function**

For 2 and 4-qubit registers respectively, this equates to the patterns,

$$\begin{aligned}\textsc{Patterns}(2) &= [0, 1, 3, 2]_{10} = [00, 01, 11, 10]_2,\\ \textsc{Patterns}(4) &= [0, 1, 3, 7, 15, 14, 12, 8]_{10}\\ &= [0000, 0001, 0011, 0111,\\ &\quad 1111, 1110, 1100, 1000]_2.\end{aligned}$$

where the subscript denotes the base of the specific pattern list. With this encoding scheme, we can show that the Hamming distances between each pattern and others in the set have a well-defined position-to-distance relationship.

As an example, let us consider a 4-element basis of tokens given by $b = \{$`up`, `down`, `left`, `right`$\}$. We define up and `down` as opposites, and so should preserve the largest Hamming distance between them. This requires mapping the tokens to either 00,11, or to 10,01 for these pairs. Similarly, we find the same procedure with the remaining tokens. In this instance, we have mapped the tokens as

$$\texttt{up} \rightarrow 00;\ \texttt{down} \rightarrow 11;\ \texttt{left} \rightarrow 01;\ \texttt{right} \rightarrow 10;$$

which preserves the relationships we have discussed earlier in 3.1.

Once again, it is worth noting that the notion of similarity is complex when considering words, as such is the concept of orthogonality. It may be argued that the up-`down` relationship has more similarities than, say, a `left-down` relationship, but for the purpose of our example this definition is sufficient. Care ought to be taken into defining inter-token relationships, requiring some domain expertise of the problem being investigated. The choice of inter-token relationship taken during preparation will influence the subsequent token mappings determined later in the process.

For our work we have deemed it sufficient to define these similarities by distance between the tokens in a text; larger distances between tokens defining a larger respective Hamming distance, and smaller distances a smaller one. We can similarly extend this method to larger datasets, though the ordering problem requires a more automated approach.

For the 4-qubit encoding scheme, we must define a strategy to map the tokens to a fully connected graph, where again the respective positions of the bit-strings reflect the Hamming distance between them, as shown in figure 1. To effectively map tokens to these bit-strings, we use the following procedure:

(a) Given the chosen basis tokens, and their positions in the text, create a graph where each basis token is a single node.
(b) Calculate the distances between all $(n^2-n)/2$ token pairings.
(c) For simplicity, choose the minimum distances between each of the pairings, and create edges with this as the given weight. As an aside, alternative methods can also be used, such as mean, median, etc.
(d) With the given fully-connected graph, find the minimum Hamiltonian cycle, and use the returned ordering to map the tokens onto the bit-strings.

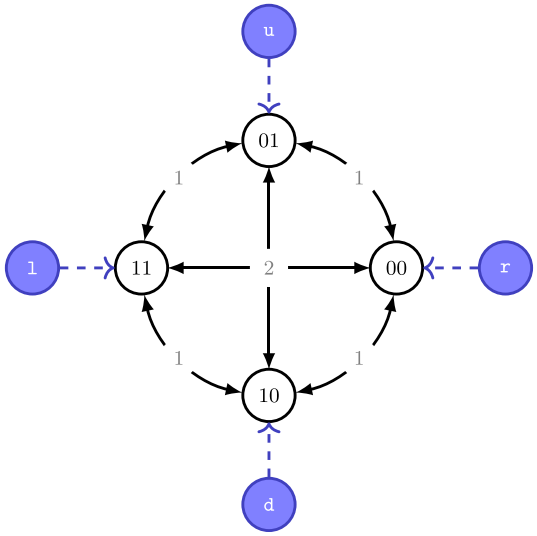

Figure 1. Graph showing the mapping of tokens (blue) to bit-strings (white). Edge weights between the bit-strings represent the Hamming distances, d_H between connected nodes. By mapping the tokens to the appropriate basis bit-string we can use the Hamming distances to represent differences between tokens.

For the calculated minimum Hamiltonian cycle, the relationships between each of the tokens will be preserved, and can effectively be mapped onto the bit-string encoding scheme. It can be noted that alternative encoding schemes and distance orderings could potentially be investigated, but will remain beyond the scope of this current work. For our purposes we make use of the `networkx` package for the finding the minimum Hamiltonian cycle [32].

3.3. Methods for quantum state encoding

To simplify our encoding procedure, we can assume a binary representation of distance for equation (1), wherein all tokens within the given cutoff are equally weighted. This allows us to encode the states as an equal-weighted superposition, and is easily implemented as a quantum circuit [33, 34].

For notational simplicity, we define the following mappings:

$$\begin{aligned} \mathrm{X}_a &: |a\rangle \rightarrow |\neg a\rangle, \\ \mathrm{CX}_{a,b} &: |a\rangle|b\rangle \rightarrow |a\rangle|a \oplus b\rangle \\ \mathrm{nCX}_{a_1 \ldots a_n b} &: |a_1\rangle \ldots |a_n\rangle|b\rangle \rightarrow \\ &\quad |a_1\rangle \ldots |a_n\rangle|b \oplus (a_1 \wedge \cdots \wedge a_n)\rangle \end{aligned}$$

where $|a\rangle$ and $|b\rangle$ are computational basis states, X is the Pauli-X (σ_x) gate, CX and nCX are the controlled X, and n-controlled NOT (nCX) operations, respectively. Additionally, we may define controlled operations using any arbitrary unitary gate using a similar construction of the above.

The goal of this algorithm is to encode a set of bit-strings representing our token meaning-space as an equal weighted superposition state. For a set of N unique binary patterns $p^{(i)} = \{p_1^{(i)}, \ldots, p_n^{(i)}\}$ each of length n for $i = 1, \ldots, N$, we require three registers of qubits; a memory register $|m\rangle$ of length n, an auxiliary register $|a\rangle$ of length n, and a control register $|u\rangle$ of length 2 initialised as $|01\rangle$. $|m\rangle$ and $|a\rangle$ are initialised as $|m\rangle = |a\rangle = |0\rangle^{\otimes n}$, with the full quantum register initialised as

$$\begin{aligned} |\psi_0\rangle &= |a\rangle|u\rangle|m\rangle \\ &= |0_1 \ldots 0_n\rangle|01\rangle|0_1 \ldots 0_n\rangle. \end{aligned} \tag{2}$$

Each of the binary vectors are encoded sequentially. For each iteration of the encoding algorithm, a new state is generated in the superposition (excluding the final iteration). The new state generated is termed as the *active* state of the next iteration. All other states are said to be *inactive*. Note, in each iteration of the algorithm, the active state will always be selected with $|u\rangle = |01\rangle$.

During a single iteration, a binary vector is stored in integer format, which is then serially encoded bit-wise with Pauli-X gates into the auxiliary register $|a\rangle$ resulting in the state $|\psi_1\rangle$:

$$|\psi_1\rangle = |a_1^{(1)} \ldots a_n^{(1)}\rangle|01\rangle|0_1 \ldots 0_n\rangle. \tag{3}$$

This binary representation is then copied into the memory register $|m\rangle$ of the active state by applying a 2CX gate on $|\psi_1\rangle$:

$$|\psi_2\rangle = \prod_{j=1}^{n} 2\mathrm{CX}_{a_j u_2 m_j} |\psi_1\rangle. \tag{4}$$

Next, we apply a CX followed by a X gate to all qubits in $|m\rangle$ using the corresponding qubits in $|a\rangle$ as controls:

$$|\psi_3\rangle = \prod_{j=1}^{n} \mathrm{X}_{m_j} \mathrm{CX}_{a_j m_j} |\psi_2\rangle. \tag{5}$$

This sets the qubits in $|m\rangle$ to 1 if the respective qubit index in both $|m\rangle$ and $|a\rangle$ match, else to 0. Thus, the state whose register $|m\rangle$ matches the pattern stored in $|a\rangle$ will be set to all 1's while the other states will have at least one occurrence of 0 in $|m\rangle$.

Now that the state being encoded has been selected, an nCX operation is applied to the first qubit in the auxiliary register using the qubits in $|m\rangle$ as the controls:

$$|\psi_4\rangle = \mathrm{nCX}_{m_1 \ldots m_n u_1} |\psi_3\rangle. \tag{6}$$

The target qubit whose initial value is 0 will be set to 1 if $|m\rangle$ consists of only 1's. This is the case for the state in the superposition whose pattern is being encoded (the current active state).

In order to populate a new state into the superposition, it is required to effectively 'carve-off' some amplitude from the existing states so the new state has a non-zero coefficient. To do this, we apply a controlled unitary matrix $\mathrm{CS}^{(N+1-i)}$ to the second auxiliary qubit u_2 using the first auxiliary qubit u_1 as a control:

$$|\psi_5\rangle = \mathrm{CS}^{(N+1-i)}_{u_1 u_2} |\psi_4\rangle,$$

where

$$\begin{aligned} \mathrm{S}^{(k)} &= \begin{bmatrix} \sqrt{\frac{k-1}{k}} & \frac{1}{\sqrt{k}} \\ -\frac{1}{\sqrt{k}} & \sqrt{\frac{k-1}{k}} \end{bmatrix} \\ &= R_y(\gamma(k)), \end{aligned} \tag{8}$$

for $k \in \{1, \ldots, N\} \subset \mathbb{Z}^+$, $R_y(\theta) = \exp(-i\theta\sigma_y/2)$ and $\gamma(k) = -\arccos((k-2)/k)$. The newly generated state will be selected with $|u\rangle = |11\rangle$, while the previous active state used to 'carve-off' this new state selected with $|u\rangle = |10\rangle$. All other states will be selected with $|u\rangle = |00\rangle$.

To apply the next iteration of the algorithm we uncompute the steps from equations (4)–(6) as

$$|\psi_6\rangle = \mathrm{nCX}_{m_1 \ldots m_n u_1} |\psi_5\rangle, \tag{9}$$

$$|\psi_7\rangle = \prod_{j=n}^{1} \mathrm{CX}_{a_j m_j} \mathrm{X}_{m_j} |\psi_6\rangle, \tag{10}$$

$$|\psi_8\rangle = \prod_{j=n}^{1} 2\mathrm{CX}_{a_j u_2 m_j} |\psi_7\rangle. \tag{11}$$

This results in the previous active state now being selected with $|u\rangle = |00\rangle$ while the new state with $|u\rangle = |01\rangle$, which identifies it as the new active state. The previous active state's memory register now contains the pattern $\{a_1^{(i)}, \ldots, a_n^{(i)}\}$ while the new active state's memory register is set to all zeroes.

Finally, the register $|a\rangle$ for every state must be set to all zeroes by sequentially applying X gates to each qubit in $|a\rangle$ according to the pattern that was just encoded. The quantum register is now ready for the next iteration to encode another pattern. Following the encoding of all patterns, our state will be

$$\begin{aligned} |\psi\rangle &= |a\rangle|u\rangle|m\rangle \\ &= |0_1 \ldots 0_n\rangle|00\rangle \left(\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |p_1^{(i)} \ldots p_n^{(i)}\rangle \right). \end{aligned} \tag{12}$$

Note, this algorithm assumes that the number of patterns to be encoded is known beforehand, which is required to generate the set of $S^{(}k)$ matrices and apply them in the correct order. The total number of qubits used in this algorithm is $2n+2$, of which $n+2$ are reusable after the implementation since the qubits in $|a\rangle$ and $|u\rangle$ are all reset to $|0\rangle$ upon completion. The additional $n+2$ qubits allows for them to be used as intermediate scratch to enable the large n-controlled operations during the encoding stages. This ensures that we can perform the nCX operations with a linear, rather than polynomial, number of two-qubit gate calls [35]. The above encoding scheme is explicitly demonstrated for both 3 and 4 qubit patterns in appendix A.

3.4. Representing patterns using encoded data

The purpose of this methodology is to represent a single test pattern using the previously encoded meaning-space. The relative distance between each meaning-space state pattern and the single test pattern $x = \{x_1, \ldots, x_n\}$ is then encoded into the amplitude of each respective meaning-space state pattern. Thus, each represented state will have a coefficient proportional to the Hamming distance between itself and the test pattern. The method we present below calculates the binary difference between the target state's bit-string and the test pattern, denoted by d_H.

The algorithm assumes that we already have N states of length n encoded into the memory register $|m\rangle$. The subsequent encoding requires $2n+1$ qubits; n qubits to store the test pattern, a single qubit register which the rotations will act on, and n qubits for the memory register. As our previously used encoding stage required $2n+2$ qubits, we can repurpose the $|a\rangle$ and $|u\rangle$ registers as the test pattern and rotation registers respectively. Our meaning-space patterns are encoded in the memory register $|m\rangle$, with registers $|a\rangle$ and $|u\rangle$ initialised as all 0's. Hence, our initial state is given by equation (12).

Next, the test pattern $x = \{x_1, \ldots, x_n\}$ is encoded into the register $|a\rangle$ sequentially by applying a X gate to each qubit whose corresponding classical bit x_i is set:

$$|\psi_1'\rangle = |x_1 \ldots x_n\rangle|00\rangle \left(\frac{1}{\sqrt{N}} \sum_{i=1}^{N} |p_1^{(i)} \ldots p_n^{(i)}\rangle \right). \tag{13}$$

Rather than overwriting register $|a\rangle$ with the differing bit-values, a two qubit controlled $R_y(\theta)$ ($2\mathrm{CR}_y^{(\theta)}$) gate is applied, such that $\theta = \frac{\pi}{n}$. This is done by iteratively applying the 2-controlled R_y gate with a_j and m_j as control qubits to rotate $|u\rangle$ if both control qubits are set for $j = 1, \ldots, n$. The operation is performed twice, such that $a_j = 1, m_j = 1$ and by appropriately flipping the bits prior to use for $a_j = 0, m_j = 0$.

Finally, the test pattern stored in register $|a\rangle$ is reset to consist of all 0's by applying a X gate to each qubit in $|a\rangle$ whose corresponding classical bit is set to 1.

The above process can be written as follows:

$$|\psi_2'\rangle = \prod_{j=1}^{n} \mathrm{X}_{a_j}\mathrm{X}_{m_j}2\mathrm{CR}_{y;a_j m_j u_1}^{\left(\frac{\pi}{n}\right)} \qquad \mathrm{X}_{a_j}\mathrm{X}_{m_j}2\mathrm{CR}_{y;a_j m_j u_1}^{\left(\frac{\pi}{n}\right)} |\psi_1\rangle, \tag{14}$$

where the state after application is given by

$$|\psi'\rangle = |0\rangle^{\otimes(n+1)} \frac{1}{\sqrt{k}} \sum_{j=1}^{k} \left[\cos\left(\phi_j\right)|0\rangle + \sin\left(\phi_j\right)|1\rangle\right] \otimes |p^{(j)}\rangle, \tag{15}$$

with

$$\phi_j = \frac{d_H(p^{(j)}, x)\pi}{n} = \frac{\pi}{n} \sum_{l=1}^{n} p_l^{(j)} \oplus x_l. \tag{16}$$

Applying the linear map

$$P = \mathbb{1}^{\otimes(n+1)} \otimes |1\rangle\langle 1| \otimes \mathbb{1}^{\otimes n}, \tag{17}$$

we represent the meaning-space states weighted by the Hamming distance with the test pattern, x. The state following this is given by

$$|\psi_x'\rangle = \frac{1}{\sqrt{k\langle\psi'|P|\psi'\rangle}} \sum_{j}^{k} \sin(\phi_j)|p_j\rangle, \tag{18}$$

where qubit registers $|a\rangle = |0\rangle^{\otimes n}$ and $|u\rangle = |01\rangle$ are left out for brevity.

With the above method we can examine the similarity between patterns mediated via the meaning space. While one may directly calculate the Hamming distance between both register states as a measure of similarity, by doing this we lose distributional meaning discussed from section 3.2. As such, we aim to represent both patterns in the meaning-space, and examine their resulting similarity using state overlap, with the result defined by

$$F(x^{(0)}, x^{(1)}) = \left| \frac{1}{k\sqrt{\langle P^{(0)}\rangle\langle P^{(1)}\rangle}} \sum_{j=1}^{k} \sin\left(\phi_j^{(0)}\right) \sin\left(\phi_j^{(1)}\right) \right|^2, \tag{19}$$

with $\langle P^{(i)}\rangle = \langle x^{(i)}|P|x^{(i)}\rangle$, and $x^{(i)}$ as test pattern i.

4. Results

4.1. Small-scale example

We now demonstrate an example of the method outlined in section 3 for a sample representation and sentence comparison problem.

We opt for the simplified `noun-verb-noun` sentence structure, and define sets of words within each of these spaces, through which we can construct our full meaning space, following the approach outlined in [4]. For nouns, we have: (i) subjects, $n_s = \{\text{ADULT}, \text{CHILD}, \text{SMITH}, \text{SURGEON}\}$; and (ii) objects, $n_o = \{\text{OUTSIDE}, \text{INSIDE}\}$. For verbs, we have $v = \{\text{STAND}, \text{SIT}, \text{MOVE}, \text{SLEEP}\}$. With these sets, we can represent the full meaning-space as given by

$$\begin{pmatrix} \text{ADULT} \\ \text{CHILD} \\ \text{SMITH} \\ \text{SURGEON} \end{pmatrix} \otimes \begin{pmatrix} \text{STAND} \\ \text{SIT} \\ \text{MOVE} \\ \text{SLEEP} \end{pmatrix} \otimes \begin{pmatrix} \text{OUTSIDE} \\ \text{INSIDE} \end{pmatrix}. \tag{20}$$

Whilst all combinations may exist, subjected to a given training corpus, only certain patterns will be observed, allowing us to restrict the information in our meaning-space. For simplicity, we can choose our corpus to be a simple set of sentences: JOHN RESTS INSIDE. MARY WALKS OUTSIDE. To represent these sentences using the bases given by equation (20), we must determine a mapping between each token in the sentences to the bases. In this instance, we manually define the mapping by taking the following meanings:

- JOHN IS AN ADULT, AND A SMITH. The state is then given as:
  $|\text{JOHN}\rangle = 1/\sqrt{2}\,(|\text{ADULT}\rangle + |\text{SMITH}\rangle)$, which is a superposition of the number of matched entities from the basis set.
- MARY IS A CHILD, AND A SURGEON. Similarly, the state is given as:
  $|\text{MARY}\rangle = 1/\sqrt{2}\,(|\text{CHILD}\rangle + |\text{SURGEON}\rangle)$, following the same procedure as above.

We also require meanings for RESTS and WALKS. If we examine synonyms for RESTS and cross-compare with our chosen vocabulary, we can find SIT and SLEEP. Similarly, for WALKS we can have STAND and MOVE. We can define the states of these words as $|\text{REST}\rangle = 1/\sqrt{2}\,(|\text{SIT}\rangle + |\text{SLEEP}\rangle)$ and $|\text{WALK}\rangle = 1/\sqrt{2}\,(|\text{STAND}\rangle + |\text{MOVE}\rangle)$. Now that we have a means to define the states in terms of our vocabulary, we can begin constructing states to encode the data.

We begin by tokenising the respective sentences into the 3 different categories: subject nouns, verbs, and object nouns. With the sentence tokenised, we next represent them as binary integers, and encode them using the processes of section 3. The basis tokens are defined in table 1. We define the mapping of 'John rests inside, Mary walks outside' to this basis in table 2.

If we consider the JOHN and MARY sentences separately for the moment, they are respectively given by the states $(1/2)|0\rangle \otimes (|10\rangle + |11\rangle) \otimes (|00\rangle + |10\rangle)$ for John, and $(1/2)|1\rangle \otimes (|00\rangle + |01\rangle) \otimes (|01\rangle + |11\rangle)$ for

Table 1. Basis data.

Dataset	Token	Bin. Index
n_s	Adult	00
n_s	Child	11
n_s	Smith	10
n_s	Surgeon	01
v	Stand	00
v	Move	01
v	Sit	11
v	Sleep	10
n_o	Inside	0
n_o	Outside	1

Table 2. Sentence data encoding using basis from table 1.

Dataset	Token	State
n_s	John	$(\|00\rangle + \|10\rangle)/\sqrt{2}$
n_s	Mary	$(\|01\rangle + \|11\rangle)/\sqrt{2}$
v	Walk	$(\|00\rangle + \|01\rangle)/\sqrt{2}$
v	Rest	$(\|10\rangle + \|11\rangle)/\sqrt{2}$
n_o	Inside	$\|0\rangle$
n_o	Outside	$\|1\rangle$

Mary. Note that we choose a little endian encoding schema, wherein the subject nouns are encoded to the right of the register and object nouns to the left. Tidying these states up yields

$$\text{John rests inside} \rightarrow |J\rangle = \frac{1}{2}(|01100\rangle + |01000\rangle + |01110\rangle + |01010\rangle),$$
$$\text{Mary walks outside} \rightarrow |M\rangle = \frac{1}{2}(|10011\rangle + |10111\rangle + |10001\rangle + |10101\rangle),$$

where the full meaning is given by $|m\rangle = \frac{|J\rangle + |M\rangle}{\sqrt{2}}$, which is a superposition of the 8 unique encodings defined by our meaning-space and sentences.

From here we will next encode a test state to be stored in register $|a\rangle$ for representation using the encoded meaning-space. We use the pattern denoted by 'Adult(s) stand inside', which is encoded as $|a\rangle = |00000\rangle$. Constructing our full state in the format of equation (13), we get

$$|\psi\rangle = \frac{1}{2\sqrt{2}}|00000\rangle \otimes |00\rangle \otimes (|01100\rangle + |01000\rangle + |01110\rangle + |01010\rangle + |10011\rangle + |10111\rangle + |10001\rangle + |10101\rangle\ ).$$

By following the steps outlined in section 3.4, rotating a single qubit from the control register $|u\rangle$ based on the Hamming distance between both registers, and applying the map from equation (17), the state of register $|m\rangle$ encodes a representation of the test pattern in the amplitude of each unique meaning-space state.

Through repeated preparation and measurement of the $|m\rangle$ register we can observe the patterns closest to the test. Figure 2 shows the observed distribution using two different patterns; adult, sit, inside (00000, orange), and child, move, inside (00111, green) compared with the encoded meaning-space patterns following equation (12) (blue). An example demonstrator of the above procedure is provided in a Jupyter notebook as part of the Github repository [11].

Given this ability to represent patterns, we can extend this approach to examine the similarity of different patterns using equation (19). One can create an additional memory register $|m'\rangle$, and perform a series of SWAP tests between both encoded patterns, to determine a measure of similarity. For the above example, we obtain an overlap of $F(00000, 00111) = 0.8602$, denoting a good degree of similarity, given our chosen meaning-space.

4.2. Automated large-scale encoding

As the previous example was artificially constructed to showcase the method, an automated workflow that determines the basis and mapped tokens, and performs the subsequent experiment is beneficial. Here we perform the same analysis, but using Lewis Carroll's 'Alice in Wonderland' in an end-to-end simulation.

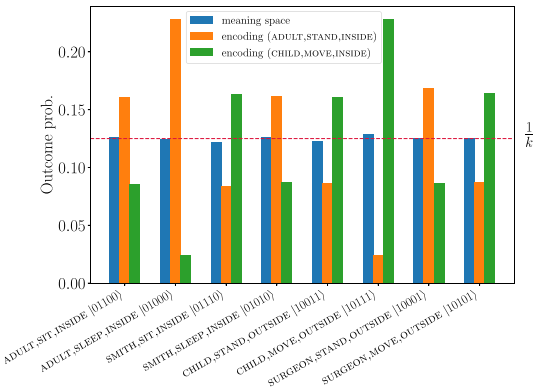

Figure 2. Sentence encoding state distribution taken by multi-shot preparation and measurement of $|m\rangle$ prior to, and post, the encoding of test patterns. Two distinct patterns are used: 00000 → (ADULT, STAND, INSIDE) (orange) and 00111 → (CHILD, MOVE, INSIDE) (green). The distribution is sampled 5×10^4 times, and shows how the Hamming distance weighting modifies the distribution relative to the $k = 8$ unweighted meaning-space states (blue).

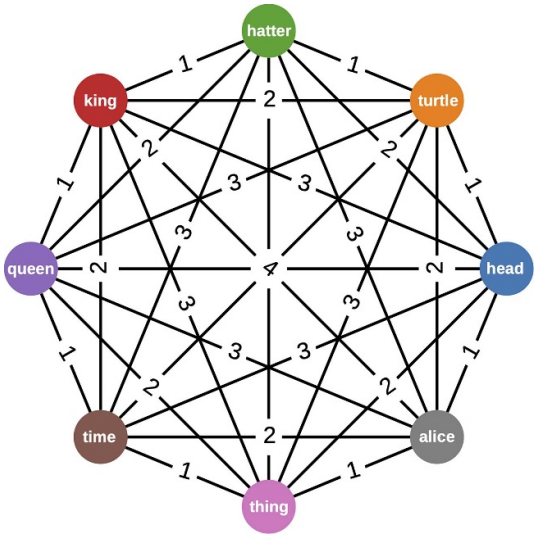

Figure 3. Relative ordering of an 8-basis set chosen for the noun dataset in 'Alice in Wonderland', using the encodings and ordering given by equation (21). The edge weight between each token shows the Hamming distance between the respective encoding patterns.

To showcase the basis choice, we will consider the nouns basis set. We define a maximum basis set of 8 nouns ($N_{\text{nouns}} = 8$), taken by their frequency of occurrence. Following the process outlined in section 3, we define a graph from these tokens, and use their inter-token distances to determine ordering following a minimum Hamiltonian cycle calculation. The resulting graph is shown by figure 3. From here we map the tokens to an appropriate set of encoding bit-strings for quantum state representation, making use of Alg. (3.2). The resulting set of mappings is

$$\begin{array}{cccccc} \text{head} & \to & |0000\rangle, & \text{turtle} & \to & |0001\rangle, \\ \text{hatter} & \to & |0011\rangle, & \text{king} & \to & |0111\rangle, \\ \text{queen} & \to & |1111\rangle, & \text{time} & \to & |1110\rangle, \\ \text{thing} & \to & |1100\rangle, & \text{alice} & \to & |1000\rangle. \end{array} \tag{21}$$

We can now map the composite tokens onto the chosen basis encoding using a distance cut-off, W_{nouns}. Following the inter-word distance calculation approach used to determine basis order, we calculate the distance between the other corpus tokens and the respective basis set. Taking the set of all nouns in the corpus as s_n, and the noun basis set as $b_n \subset s_n$, for every token t_n in s_n we perform

$$t_n : s_n \mapsto b_n.$$

Tokens that fall outside W_{nouns} are mapped to the empty set, $\emptyset$. This approach is then repeated for verbs, and lastly inter-dataset distances between noun-verb pairings, W_{nv}, which are used to discover viable sentences.

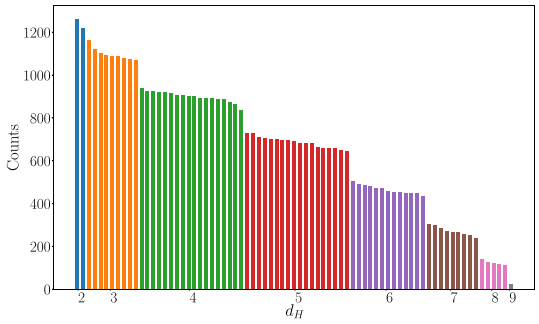

Figure 4. The result of measuring $|m\rangle$ states following the encoding of the string pattern 'hatter say queen'. This example uses a 75 unique pattern basis set, and taking 5×10^4 samples to build the distribution. The Hamming distances of the labels are indicated on the x-axis, and differentiated by colour, where we can see clear distinction between the patterns in each Hamming category. The data mapping tokens to patterns in each category can be viewed in appendix D.

The mapped composite tokens may then be used to create a compositional sentence structure by tensoring the respective token states.

Following the previous example, we may examine the automatic encoding and representation of the string 'hatter say queen' to the meaning-space patterns. Given that representing the text in its entirety would be a substantial challenge, we limit the amount of information to be encoded by controlling the pre-processing steps as $N_{\text{nouns}} = 8$, $N_{\text{verbs}} = 4$, $W_{\text{nouns}} = 5$, $W_{\text{verbs}} = 5$ and $W_{\text{vn}} = 4$. Here N_{nouns} is again the number of basis nouns in both subject and object datasets, N_{verbs} the number of basis verbs, W_{nouns} and W_{verbs} the cutoff distances for mapping other nouns and verbs in the corpus nouns to the basis tokens, and W_{vn} is the cutoff distance to relate noun and verb tokens.

For the above parameters, the method finds a subset of 75 unique patterns to represent the corpus. Following section 4.1 one obtains the associated similarity of encoded elements by the resulting likelihood of occurrence, as indicated by figure 4, where we have prepared and sampled the $|m\rangle$ register 5×10^4 times to build the distribution. Clear step-wise distinctions can be observed between the different categories of Hamming-weighted states, with the full list presented in appendix D table D1. Given the basis encoding tokens from equation (21), the string 'hatter say queen' can be mapped to the value 995 (`1111100011` in binary).

As before, we can also compare patterns mediated via the meaning-space. For the pattern 'hatter say queen', the most similar patterns are 'hatter say king' (`0111100011`), 'hatter go queen' (`1111110011`) and 'turtle say queen' (`1111100001`) with overlaps of 0.974, 0.974 and 0.973 respectively. We include a variety of other encoded comparisons in the appendix as table E2 to showcase the method.

5. Conclusions

In this paper we have demonstrated methods for encoding corpus data as quantum states. Taking elements from the categorical distributional compositional semantic formalism, we developed a proof-of-concept workflow for preparing small and large scale data sets to be encoded, processed, and decoded using a given quantum register. We showed the preparation, encoding, comparison, and decoding of small and large datasets using the presented methods.

Recent works have shown the importance of the reduction in classical data to be represented on a quantum system [36]. The approach defined above follows an analogous procedure, representing the important elements of the corpus data using a fundamental subset of the full corpus data. Using this subset, we have shown how to represent meanings, and subsequently the calculation of similarity between different meaning representations. We have additionally released all of this work as part of an Apache licensed open-source toolkit [11].

For completeness, it is worth mentioning the circuit depths required to realise the above procedures. Taking the large scale example, we obtain single and two-qubit gate call counts of 2413 and 33175 respectively to encode the meaning space. This may be difficult to realise on current NISQ generation quantum systems, where the use of simulators instead allow us to make gains in understanding of applying these methods to real datasets.

The potential for circuit optimisation through the use of ZX calculus [37], or circuit compilation through tools such as CQC's t|ket⟩ may offer more realistic circuit depths, especially when considering mapping to physical qubit register topologies [38].

Very recent works on the implementation of the DisCoCat formalism on physical devices without the need for qRAM, have also emerged [39]. These methods may provide a more generalised approach to investigate quantum machine learning models in NLP and beyond, and have the potential to overcome the limitations discussed earlier with data encoding. We imagine the merging of this generalised approach [40] with the hybrid quantum–classical methods we have devised to allow interesting results and further development of this field. We leave this to future work.

Acknowledgments

We would like to thank Prof. Bob Coecke and Dr Ross Duncan for discussions and suggestions during the early stages of this work. The work leading to this publication has received funding from Enterprise Ireland and the European Union's Regional Development Fund under grant agreement IP 2018 0751. The opinions, findings and conclusions or recommendations expressed in this material are those of the authors and neither Enterprise Ireland nor the European Union are liable for any use that may be made of information contained herein. The authors also acknowledge funding and support from Intel during the duration of this project.

Appendix A. Encoding examples

A.1. 3 qubits, 4 patterns

Starting with a $n = 3$ qubit register, we aim to encode the following $N = 4$ patterns:

$$P = [001, 111, 000, 110],$$

which are taken from the set of available patterns from Alg. 3.2 for a 3 qubit data-set. We follow the encoding procedure as defined by section 3.2, with the initial state defined by equation (2) as

$$\begin{aligned}|\Psi_0\rangle &= |a\rangle|u\rangle|m\rangle \\ &= |a_1a_2a_3\rangle|u_1u_2\rangle|m_1m_2m_3\rangle \\ &= |000\rangle|01\rangle|000\rangle.\end{aligned}$$

Starting with $i = 1$, where i denotes the index of the pattern ($i \in \{1, 2, 3, 4\}$), we aim to encode pattern 001. We begin by directly encoding our first binary pattern to the auxiliary register $|a\rangle$ by applying Pauli-X gates for the classical bits that are set:

$$|\Psi_1\rangle = |001\rangle|01\rangle|000\rangle.$$

Next, we copy the state stored in $|a\rangle$ to the memory register $|m\rangle$ for the active state (i.e. state with control register $|u\rangle = |01\rangle$), giving

$$\begin{aligned}|\Psi_2\rangle &= \prod_{j=1}^{n} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_1\rangle \\ &= (2\mathrm{CX}_{a_3,u_2,m_3})(2\mathrm{CX}_{a_2,u_2,m_2})(2\mathrm{CX}_{a_1,u_2,m_1})|001\rangle|01\rangle|000\rangle \\ &= |001\rangle|01\rangle|001\rangle.\end{aligned}$$

We then set the qubits of each state in $|m\rangle$ to 1 if the corresponding qubit index in $|a\rangle$ matches that of the same index in $|m\rangle$. This results in the active state $|m\rangle$ set to all 1's:

$$\begin{aligned}|\Psi_3\rangle &= \prod_{j=1}^{n} \mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi_2\rangle \\ &= (\mathrm{X}_{m_3}\mathrm{CX}_{a_3,m_3})(\mathrm{X}_{m_2}\mathrm{CX}_{a_2,m_2})(\mathrm{X}_{m_1}\mathrm{CX}_{a_1,m_1})|001\rangle|01\rangle|001\rangle \\ &= |001\rangle|01\rangle|111\rangle,\end{aligned}$$

Next, the active state has $|u\rangle$ set to $|11\rangle$:

$$\begin{aligned}|\Psi_4\rangle &= n\mathrm{CX}_{m_1,\ldots,m_n,u_1}|\Psi_3\rangle \\ &= 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|001\rangle|01\rangle|111\rangle \\ &= |001\rangle|11\rangle|111\rangle.\end{aligned}$$

A new state is 'carved off' from the current active state. This new state is then considered the 'active state' for the next iteration:

$$|\Psi_5\rangle = \mathrm{CS}_{u_1,u_2}^{(N+1-i)}|\Psi_4\rangle,$$

where we require use of equation (8), defined by

$$\mathrm{S}^{(k)} = R_y\left(-\cos^{-1}\left(\frac{k-2}{k}\right)\right).$$

As we intend to encode 4 patterns, we require 4 such matrices, defined by the parameters $\mathrm{S}^{(N+1-i)}$; we then create matrices for $\mathrm{S}^{(4)}$, $\mathrm{S}^{(3)}$, $\mathrm{S}^{(2)}$ and $\mathrm{S}^{(1)}$ as

$$\mathrm{S}^{(4)} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix},\ \mathrm{S}^{(3)} = \begin{bmatrix} \sqrt{\frac{2}{3}} & \frac{1}{\sqrt{3}} \\ \frac{-1}{\sqrt{3}} & \sqrt{\frac{2}{3}} \end{bmatrix},\ \mathrm{S}^{(2)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix},\ \mathrm{S}^{(1)} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

Given that we are currently encoding the pattern at index $i = 1$, we use the $\mathrm{S}^{(4)}$ matrix here

$$\begin{aligned}|\Psi_5\rangle &= \mathrm{CS}_{u_1,u_2}^{(N+1-i)}|\Psi_4\rangle \\ &= \mathrm{CS}_{u_1,u_2}^{(4)}|\Psi_4\rangle \\ &= \frac{1}{2}|001\rangle|10\rangle|111\rangle + \frac{\sqrt{3}}{2}|001\rangle|11\rangle|111\rangle.\end{aligned}$$

We now must uncompute our previous operations to repeat the process.

$$\begin{aligned}|\Psi_6\rangle &= 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|\Psi_5\rangle. \\ &= \frac{1}{2}|001\rangle|00\rangle|111\rangle + \frac{\sqrt{3}}{2}|001\rangle|01\rangle|111\rangle.\end{aligned}$$

From above, we can see that we have split off the encoded pattern, which is currently selected with $|u\rangle = |00\rangle$. The memory register selected with $|u\rangle = |01\rangle$ forms the new resource for encoding the next pattern, following completion of the unencoding steps:

$$\begin{aligned}|\Psi_7\rangle &= \prod_{j=n}^{1}\mathrm{CX}_{a_j,m_j}\mathrm{X}_{m_j}|\Psi_6\rangle \\ &= \frac{1}{2}|001\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|001\rangle|01\rangle|001\rangle,\end{aligned}$$

$$\begin{aligned}|\Psi_8\rangle &= \prod_{j=n}^{1}2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7\rangle \\ &= \frac{1}{2}|001\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|001\rangle|01\rangle|000\rangle.\end{aligned}$$

We now reset register $|a\rangle$ and start fresh for the pattern at index $i = 2$, 111:

$$|\Psi_0'\rangle = \frac{1}{2}|000\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|000\rangle|01\rangle|000\rangle,$$

where following the same operations and steps as above, we get

$$|\Psi_1'\rangle = \frac{1}{2}|111\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|111\rangle|01\rangle|000\rangle,$$

$$|\Psi_2'\rangle = \prod_{j=1}^{n} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_1'\rangle$$
$$= \frac{1}{2}|111\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|111\rangle|01\rangle|111\rangle,$$

$$|\Psi_3'\rangle = \prod_{j=1}^{n} \mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi_2'\rangle$$
$$= \frac{1}{2}|111\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|111\rangle|01\rangle|111\rangle,$$

$$|\Psi_4'\rangle = 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|\Psi_3'\rangle$$
$$= \frac{1}{2}|111\rangle|00\rangle|001\rangle + \frac{\sqrt{3}}{2}|111\rangle|11\rangle|111\rangle,$$

$$|\Psi_5'\rangle = \mathrm{CS}_{u_1,u_2}^{(N+1-i)}|\Psi_4'\rangle$$
$$= \mathrm{CS}_{u_1,u_2}^{(3)}|\Psi_4'\rangle$$
$$= \frac{1}{2}\left(|111\rangle|00\rangle|001\rangle + |111\rangle|10\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|111\rangle|11\rangle|111\rangle,$$

$$|\Psi_6'\rangle = 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|\Psi_5'\rangle$$
$$= \frac{1}{2}\left(|111\rangle|00\rangle|001\rangle + |111\rangle|00\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|111\rangle|01\rangle|111\rangle,$$

$$|\Psi_7'\rangle = \prod_{j=n}^{1} \mathrm{CX}_{a_j,m_j}\mathrm{X}_{m_j}|\Psi_6'\rangle$$
$$= \frac{1}{2}\left(|111\rangle|00\rangle|001\rangle + |111\rangle|00\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|111\rangle|01\rangle|111\rangle,$$

$$|\Psi_8'\rangle = \prod_{j=n}^{1} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7'\rangle$$
$$= \frac{1}{2}\left(|111\rangle|00\rangle|001\rangle + |111\rangle|00\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|111\rangle|01\rangle|000\rangle.$$

As before, we reset register $|a\rangle$ for pattern at index $i = 3\,000$ giving

$$|\Psi_0''\rangle = \frac{1}{2}\left(|000\rangle|00\rangle|001\rangle + |000\rangle|00\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|000\rangle|01\rangle|000\rangle,$$

$$|\Psi_1''\rangle = \frac{1}{2}\left(|000\rangle|00\rangle|001\rangle + |000\rangle|00\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|000\rangle|01\rangle|000\rangle,$$

$$|\Psi_2''\rangle = \prod_{j=1}^{n} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_1''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|001\rangle + |000\rangle|00\rangle|111\rangle\right) + \frac{1}{\sqrt{2}}|000\rangle|01\rangle|000\rangle,$$

$$|\Psi_3''\rangle = \prod_{j=1}^{n} \mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi_2''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|110\rangle + |000\rangle|00\rangle|000\rangle\right) + \frac{1}{\sqrt{2}}|000\rangle|01\rangle|111\rangle,$$

$$|\Psi_4''\rangle = 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|\Psi_3''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|110\rangle + |000\rangle|00\rangle|000\rangle\right) + \frac{1}{\sqrt{2}}|000\rangle|11\rangle|111\rangle,$$

$$|\Psi_5''\rangle = \mathrm{CS}^{(2)}_{u_1,u_2}|\Psi_4''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|110\rangle + |000\rangle|00\rangle|000\rangle + |000\rangle|10\rangle|111\rangle + |000\rangle|11\rangle|111\rangle\right),$$

$$|\Psi_6''\rangle = 3\mathrm{CS}_{m_1,m_2,m_3,u_1}|\Psi_5''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|110\rangle + |000\rangle|00\rangle|000\rangle + |000\rangle|00\rangle|111\rangle + |000\rangle|01\rangle|111\rangle\right),$$

$$|\Psi_7''\rangle = \prod_{j=n}^{1}\mathrm{CX}_{a_j,m_j}X_{m_j}|\Psi_6''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|001\rangle + |000\rangle|00\rangle|111\rangle + |000\rangle|00\rangle|000\rangle + |000\rangle|01\rangle|000\rangle\right),$$

$$|\Psi_8''\rangle = \prod_{j=n}^{1}2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7''\rangle$$
$$= \frac{1}{2}\left(|000\rangle|00\rangle|001\rangle + |000\rangle|00\rangle|111\rangle + |000\rangle|00\rangle|000\rangle + |000\rangle|01\rangle|000\rangle\right).$$

Finally, we encode the last pattern at index $i = 4$, 110 giving

$$|\Psi_0'''\rangle = \frac{1}{2}\left(|000\rangle|00\rangle|001\rangle + |000\rangle|00\rangle|111\rangle + |000\rangle|00\rangle|000\rangle + |000\rangle|01\rangle|000\rangle\right),$$

$$|\Psi_1'''\rangle = \frac{1}{2}\left(|110\rangle|00\rangle|001\rangle + |110\rangle|00\rangle|111\rangle + |110\rangle|00\rangle|000\rangle + |110\rangle|01\rangle|000\rangle\right),$$

$$|\Psi_2'''\rangle = \prod_{j=1}^{n}2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_1'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|001\rangle + |110\rangle|00\rangle|111\rangle + |110\rangle|00\rangle|000\rangle + |110\rangle|01\rangle|110\rangle\right),$$

$$|\Psi_3'''\rangle = \prod_{j=1}^{n}\mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi_2'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|000\rangle + |110\rangle|00\rangle|110\rangle + |110\rangle|00\rangle|001\rangle + |110\rangle|01\rangle|111\rangle\right),$$

$$|\Psi_4'''\rangle = 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|\Psi_3'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|000\rangle + |110\rangle|00\rangle|110\rangle + |110\rangle|00\rangle|001\rangle + |110\rangle|11\rangle|111\rangle\right),$$

$$|\Psi_5'''\rangle = \mathrm{CS}^{(1)}_{u_1,u_2}|\Psi_4'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|000\rangle + |110\rangle|00\rangle|110\rangle + |110\rangle|00\rangle|001\rangle + |110\rangle|10\rangle|111\rangle\right),$$

$$|\Psi_6'''\rangle = 3\mathrm{CX}_{m_1,m_2,m_3,u_1}|\Psi_5'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|000\rangle + |110\rangle|00\rangle|110\rangle + |110\rangle|00\rangle|001\rangle + |110\rangle|00\rangle|111\rangle\right),$$

$$|\Psi_7'''\rangle = \prod_{j=n}^{1}\mathrm{CX}_{a_j,m_j}\mathrm{X}_{m_j}|\Psi_6'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|001\rangle + |110\rangle|00\rangle|111\rangle + |110\rangle|00\rangle|000\rangle + |110\rangle|00\rangle|110\rangle\right),$$

$$|\Psi_8'''\rangle = \prod_{j=n}^{1} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7'''\rangle$$
$$= \frac{1}{2}\left(|110\rangle|00\rangle|001\rangle + |110\rangle|00\rangle|111\rangle + |110\rangle|00\rangle|000\rangle + |110\rangle|00\rangle|110\rangle\right).$$

After unencoding the pattern from $|a\rangle$, our final encoded state is

$$|\Psi\rangle = \frac{1}{2}|0\rangle^{\otimes 5}\left(|001\rangle + |111\rangle + |000\rangle + |110\rangle\right).$$

In register $|m\rangle$, we now have the bit-patterns encoded with equal amplitudes, and completely separable from registers $|a\rangle$ and $|u\rangle$.

A.2. 4 qubits, 3 patterns

As with App. A.1, we follow the encoding procedure from section 3.2. Starting with a $n = 4$ qubit register, we aim to encode the following $N = 3$ patterns:

$$P = [1111, 0001, 1100],$$

which are again taken from the set of available patterns from Alg. 3.2 for a 4 qubit data-set, beginning with state

$$|\Psi_0\rangle = |a\rangle|u\rangle|m\rangle$$
$$= |a_1a_2a_3a_4\rangle|u_1u_2\rangle|m_1m_2m_3m_4\rangle$$
$$= |0000\rangle|01\rangle|0000\rangle.$$

Starting with index $i = 1$, we will begin with pattern 1111.

$$|\Psi_1\rangle = |1111\rangle|01\rangle|0000\rangle,$$

$$|\Psi_2\rangle = \prod_{j=1}^{n} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_1\rangle$$
$$= (2\mathrm{CX}_{a_4,u_2,m_4})(2\mathrm{CX}_{a_3,u_2,m_3})(2\mathrm{CX}_{a_2,u_2,m_2})(2\mathrm{CX}_{a_1,u_2,m_1})|1111\rangle|01\rangle|0000\rangle$$
$$= |1111\rangle|01\rangle|1111\rangle,$$

$$|\Psi_3\rangle = \prod_{j=1}^{n} \mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi_2\rangle$$
$$= (\mathrm{X}_{m_4}\mathrm{CX}_{a_4,m_4})(\mathrm{X}_{m_3}\mathrm{CX}_{a_3,m_3})(\mathrm{X}_{m_2}\mathrm{CX}_{a_2,m_2})(\mathrm{X}_{m_1}\mathrm{CX}_{a_1,m_1})|1111\rangle|01\rangle|1111\rangle$$
$$= |1111\rangle|01\rangle|1111\rangle,$$

$$|\Psi_4\rangle = n\mathrm{CX}_{m_1,\ldots,m_n,u_1}|\Psi_3\rangle$$
$$= 4\mathrm{CX}_{m_1,m_2,m_3,m_4,u_1}|1111\rangle|01\rangle|1111\rangle$$
$$= |1111\rangle|11\rangle|1111\rangle,$$

$$|\Psi_5\rangle = \mathrm{CS}_{u_1,u_2}^{(N+1-i)}|\Psi_4\rangle,$$

where we again require use of equation (8). Given that we intend to encode 3 patterns this time, we require 3 such matrices, $\mathrm{S}^{(3)}$, $\mathrm{S}^{(2)}$ and $\mathrm{S}^{(1)}$ as

$$\mathrm{S}^{(3)} = \begin{bmatrix} \sqrt{\frac{2}{3}} & \frac{1}{\sqrt{3}} \\ \frac{-1}{\sqrt{3}} & \sqrt{\frac{2}{3}} \end{bmatrix}, \mathrm{S}^{(2)} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \mathrm{S}^{(1)} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}.$$

For index $i=1$, we use the $S^{(3)}$ matrix:

$$\begin{aligned}|\Psi_5\rangle &= \mathrm{CS}^{(N+1-i)}_{u_1,u_2}|\Psi_4\rangle \\ &= \mathrm{CS}^{(3)}_{u_1,u_2}|\Psi_4\rangle, \\ &= \frac{1}{\sqrt{3}}|1111\rangle|10\rangle|1111\rangle + \sqrt{\frac{2}{3}}|1111\rangle|11\rangle|1111\rangle,\end{aligned}$$

$$\begin{aligned}|\Psi_6\rangle &= 4\mathrm{CX}_{m_1,m_2,m_3,m_4,u_1}|\Psi_5\rangle \\ &= \frac{1}{\sqrt{3}}|1111\rangle|00\rangle|1111\rangle + \sqrt{\frac{2}{3}}|1111\rangle|01\rangle|1111\rangle,\end{aligned}$$

$$\begin{aligned}|\Psi_7\rangle &= \prod_{j=n}^{1}\mathrm{CX}_{a_j,m_j}\mathrm{X}_{m_j}|\Psi_6\rangle \\ &= \frac{1}{\sqrt{3}}|1111\rangle|00\rangle|1111\rangle + \sqrt{\frac{2}{3}}|1111\rangle|01\rangle|1111\rangle,\end{aligned}$$

$$\begin{aligned}|\Psi_8\rangle &= \prod_{j=n}^{1}2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7\rangle \\ &= \frac{1}{\sqrt{3}}|1111\rangle|00\rangle|1111\rangle + \sqrt{\frac{2}{3}}|1111\rangle|01\rangle|0000\rangle.\end{aligned}$$

Resetting the register $|a\rangle$ for index $i=2$, pattern 0001 gives

$$|\Psi'_0\rangle = \frac{1}{\sqrt{3}}|0000\rangle|00\rangle|1111\rangle + \sqrt{\frac{2}{3}}|0000\rangle|01\rangle|0000\rangle,$$

$$|\Psi'_1\rangle = \frac{1}{\sqrt{3}}|0001\rangle|00\rangle|1111\rangle + \sqrt{\frac{2}{3}}|0001\rangle|01\rangle|0000\rangle,$$

$$\begin{aligned}|\Psi'_2\rangle &= \prod_{j=1}^{n}2\mathrm{CX}_{a_j,u_2,m_j}|\Psi'_1\rangle \\ &= \frac{1}{\sqrt{3}}|0001\rangle|00\rangle|1111\rangle + \sqrt{\frac{2}{3}}|0001\rangle|01\rangle|0001\rangle,\end{aligned}$$

$$\begin{aligned}|\Psi'_3\rangle &= \prod_{j=1}^{n}\mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi'_2\rangle \\ &= \frac{1}{\sqrt{3}}|0001\rangle|00\rangle|0001\rangle + \sqrt{\frac{2}{3}}|0001\rangle|01\rangle|1111\rangle,\end{aligned}$$

$$\begin{aligned}|\Psi'_4\rangle &= 4\mathrm{CX}_{m_1,m_2,m_3,m_4,u_1}|\Psi'_3\rangle \\ &= \frac{1}{\sqrt{3}}|0001\rangle|00\rangle|0001\rangle + \sqrt{\frac{2}{3}}|0001\rangle|11\rangle|1111\rangle.\end{aligned}$$

For $i=2$, we use the matrix $S^{(2)}$:

$$\begin{aligned}|\Psi'_5\rangle &= \mathrm{CS}^{(N+1-i)}_{u_1,u_2}|\Psi'_4\rangle \\ &= \mathrm{CS}^{(2)}_{u_1,u_2}|\Psi'_4\rangle \\ &= \frac{1}{\sqrt{3}}\left(|0001\rangle|00\rangle|0001\rangle + |0001\rangle|10\rangle|1111\rangle + |0001\rangle|11\rangle|1111\rangle\right),\end{aligned}$$

$$\begin{aligned}|\Psi'_6\rangle &= 4\mathrm{CX}_{m_1,m_2,m_3,m_4,u_1}|\Psi'_5\rangle \\ &= \frac{1}{\sqrt{3}}\left(|0001\rangle|00\rangle|0001\rangle + |0001\rangle|00\rangle|1111\rangle + |0001\rangle|01\rangle|1111\rangle\right),\end{aligned}$$

$$|\Psi_7'\rangle = \prod_{j=n}^{1} \mathrm{CX}_{a_j,m_j}\mathrm{X}_{m_j}|\Psi_6'\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|0001\rangle|00\rangle|1111\rangle + |0001\rangle|00\rangle|0001\rangle + |0001\rangle|01\rangle|0001\rangle\right),$$

$$|\Psi_8'\rangle = \prod_{j=n}^{1} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7'\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|0001\rangle|00\rangle|1111\rangle + |0001\rangle|00\rangle|0001\rangle + |0001\rangle|01\rangle|0000\rangle\right).$$

For the final pattern, we reset $|a\rangle$ for index $i = 3$, pattern 1100, giving:

$$|\Psi_0''\rangle = \frac{1}{\sqrt{3}}\left(|0000\rangle|00\rangle|1111\rangle + |0000\rangle|00\rangle|0001\rangle + |0000\rangle|01\rangle|0000\rangle\right),$$

$$|\Psi_1''\rangle = \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1111\rangle + |1100\rangle|00\rangle|0001\rangle + |1100\rangle|01\rangle|0000\rangle\right),$$

$$|\Psi_2''\rangle = \prod_{j=1}^{n} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_1''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1111\rangle + |1100\rangle|00\rangle|0001\rangle + |1100\rangle|01\rangle|1100\rangle\right),$$

$$|\Psi_3''\rangle = \prod_{j=1}^{n} \mathrm{X}_{m_j}\mathrm{CX}_{a_j,m_j}|\Psi_2''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1100\rangle + |1100\rangle|00\rangle|0010\rangle + |1100\rangle|01\rangle|1111\rangle\right),$$

$$|\Psi_4''\rangle = 4\mathrm{CX}_{m_1,m_2,m_3,m_4,u_1}|\Psi_3''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1100\rangle + |1100\rangle|00\rangle|0010\rangle + |1100\rangle|11\rangle|1111\rangle\right).$$

We now use matrix $\mathrm{S}^{(1)}$:

$$|\Psi_5''\rangle = \mathrm{CS}^{(1)}_{u_1,u_2}|\Psi_4''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1100\rangle + |1100\rangle|00\rangle|0010\rangle + |1100\rangle|10\rangle|1111\rangle\right),$$

$$|\Psi_6''\rangle = 4\mathrm{CX}_{m_1,m_2,m_3,m_4,u_1}|\Psi_5''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1100\rangle + |1100\rangle|00\rangle|0010\rangle + |1100\rangle|00\rangle|1111\rangle\right),$$

$$|\Psi_7''\rangle = \prod_{j=n}^{1} \mathrm{CX}_{a_j,m_j}\mathrm{X}_{m_j}|\Psi_6''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1111\rangle + |1100\rangle|00\rangle|0001\rangle + |1100\rangle|00\rangle|1100\rangle\right),$$

$$|\Psi_8''\rangle = \prod_{j=n}^{1} 2\mathrm{CX}_{a_j,u_2,m_j}|\Psi_7''\rangle$$
$$= \frac{1}{\sqrt{3}}\left(|1100\rangle|00\rangle|1111\rangle + |1100\rangle|00\rangle|0001\rangle + |1100\rangle|00\rangle|1100\rangle\right).$$

After unencoding the pattern from $|a\rangle$, our final encoded state is:

$$|\Psi\rangle = \frac{1}{\sqrt{3}}|0\rangle^{\otimes 6}\left(|1111\rangle + |0001\rangle + |1100\rangle\right).$$

As with App. A.1, we now have the bit-patterns encoded with equal amplitudes, and completely separable from registers $|a\rangle$ and $|u\rangle$.

Appendix B. Corpus preparation

Our QNLP software solution [11] can target most corpora provided that adequate pre-processing is conducted prior to the main routines of the application, and follows the outline approach from section 3.

This approach has several variables that can be adjusted to control the operation of the pre-processing stage. The limiting number of top N_{nouns} and N_{verbs} are defined with the run-time parameters `NUM_BASIS_NOUN` and `NUM_BASIS_VERB`, and defined as environment variables. The number of neighbouring nouns, W_{nouns}, and verbs, W_{verbs}, to consider when mapping the corpus tokens to basis tokens, are controlled by the run-time parameters `BASIS_NOUN_DIST_CUTOFF` and `BASIS_VERB_DIST_CUTOFF` respectively, and again defined as environment variables.

Finally, for forming *noun-verb-noun* sentence structures, the number of neighbouring nouns to consider for determining the basis verbs, W_{vn}, are controlled through the environment variable `VERB_NOUN_DIST_CUTOFF`. Additionally, the sentence is only valid if the inter-noun distance on a *noun-verb-noun* structure is within $2W_{\text{vn}}$. To choose appropriate values for these parameters, one must consider the overall complexity of the corpus, the number of *noun-verb-noun* sentences, the available qubit resources, and the intended detail in representing the overall meaning. For the simplified example in section 4.1, we have a somewhat sparsely encoded set of patterns in the meaning space (8 patterns out of a possible 32), with a small number of qubits to represent the processing and assist with the encoding. A more complex text, with a larger basis set will require substantially more resources. For example, choosing `NUM_BASIS_NOUN = 10` and `NUM_BASIS_VERB = 10` using the discussed simplified cyclic encoding from algorithm (1) will require at least 32 qubits. However, depending on the amount of information the pre-processing stage can extract, this may be an overestimate or underestimate of the required resources.

Appendix C. Software dependencies

All results in this manuscript were generated using our QNLP toolkit, which is available at [11]. Jupyter notebooks, packages and scripts exist for all operations described. We made use of the Intel Quantum Simulator [12] to perform all quantum gate-level simulations, running on Kay, the Irish national supercomputer. To integrate our C++ work with Python we have made use of the pybind11 suite [41]. All results obtained were through compilation with IntelR Parallel Studio XE 2019 Update 5 for distributed workloads (section 4.2), and GCC 9.2 for shared (OpenMP) workloads (section 4.1).

To analyse and prepare the corpus data for encoding into the quantum state-space, we have used the well-defined classical routines for corpus tokenisation and tagging from the NLTK [15] and spaCy [16] software suites. For plotting we explicitly used pgfplots/tikz for figure 1, and Matplotlib for all others [42]. We additionally used the Scipy ecosystem and pandas for both the results analysis and preprocessing stages [43–45].

Appendix D. Encoded meaning-space data

Table D1 is used to generate figure 4. It encodes data from 'Alice in Wonderland' using the preprocessing control parameters:

- Number of basis elements for state encoding: `NUM_BASIS_NOUN = 8 NUM_BASIS_VERB = 4`
- Inter-token composite representation distance: `BASIS_NOUN_DIST_CUTOFF = 5, BASIS_VERB_DIST_CUTOFF = 5`
- Verb-noun distance cut-off for association: `VERB_NOUN_DIST_CUTOFF = 4`

Table D1. Results for figure 3 taking 5×10^4 samples encoding AIW using the parameters `NUM_BASIS_NOUN: 8`, `NUM_BASIS_VERB: 4`, `BASIS_NOUN_DIST_CUTOFF: 5`, `BASIS_VERB_DIST_CUTOFF: 5`, `VERB_NOUN_DIST_CUTOFF: 4`, and comparing with the test pattern ('hatter,says,queen') with binary string 1111100011.

Label	Bin. pattern	d_H	Count
king,go,queen	1111110111	2	1260
king,say,time	1110100111	2	1217
time,say,queen	1111101110	3	1165
king,go,time	1110110111	3	1123
queen,go,queen	1111111111	3	1102
hatter,say,alice	1000100011	3	1094
king,would,time	1110000111	3	1088
king,say,hatter	0011100111	3	1087
king,go,king	0111110111	3	1080
head,go,queen	1111110000	3	1075
queen,say,time	1110101111	3	1069
alice,say,king	0111101000	4	940
queen,go,king	0111111111	4	925
alice,go,queen	1111111000	4	924
head,go,king	0111110000	4	922
king,go,hatter	0011110111	4	919
time,go,queen	1111111110	4	913
king,would,hatter	0011000111	4	908
queen,would,time	1110001111	4	908
alice,say,time	1110101000	4	899
time,would,queen	1111001110	4	899
time,say,king	0111101110	4	894
king,say,alice	1000100111	4	894
time,say,time	1110101110	4	893
queen,say,hatter	0011101111	4	889
hatter,would,alice	1000000011	4	885
queen,go,time	1110111111	4	872
king,think,time	1110010111	4	866
hatter,go,alice	1000110011	4	836
thing,say,time	1110101100	5	730
king,think,hatter	0011010111	5	729
queen,say,alice	1000101111	5	712
queen,think,time	1110011111	5	704
time,go,time	1110111110	5	702
time,would,king	0111001110	5	700
alice,go,king	0111111000	5	698
time,would,time	1110001110	5	696
alice,go,time	1110111000	5	689
alice,say,hatter	0011101000	5	681
hatter,think,alice	1000010011	5	681
queen,go,hatter	0011111111	5	680
time,go,king	0111111110	5	662
queen,would,hatter	0011001111	5	658
king,would,alice	1000000111	5	657
thing,go,queen	1111111100	5	657
king,go,alice	1000110111	5	650
alice,would,time	1110001000	5	642
queen,think,hatter	0011011111	6	504
alice,think,time	1110011000	6	490
head,go,alice	1000110000	6	485
thing,would,time	1110001100	6	479
alice,go,hatter	0011111000	6	472
queen,would,alice	1000001111	6	470
alice,say,alice	1000101000	6	456
time,say,alice	1000101110	6	455
thing,go,time	1110111100	6	455
king,think,alice	1000010111	6	449
queen,go,alice	1000111111	6	449
alice,would,hatter	0011001000	6	448

(Continued)

Table D1. (Continued).

Label	Bin. pattern	d_H	Count
thing,go,king	0111111100	6	436
time,would,alice	1000001110	7	303
alice,go,alice	1000111000	7	297
alice,say,head	0000101000	7	283
time,go,alice	1000111110	7	269
alice,would,alice	1000001000	7	267
thing,say,alice	1000101100	7	266
queen,think,alice	1000011111	7	259
time,say,head	0000101110	7	254
alice,think,hatter	0011011000	7	239
thing,go,alice	1000111100	8	139
thing,would,alice	1000001100	8	125
alice,think,alice	1000011000	8	123
time,would,head	0000001110	8	117
time,go,head	0000111110	8	113
alice,think,head	0000011000	9	24

Appendix E. Overlap comparison data

Table E1 presents comparison data for the basis-token composed sentence 'Hatter say Queen' and and a variety of other allowed sentence structures. Data is again encoded from 'Alice in Wonderland' using the preprocessing control parameters

- Number of basis elements for state encoding: `NUM_BASIS_NOUN = 8 NUM_BASIS_VERB = 4`
- Inter-token composite representation distance: `BASIS_NOUN_DIST_CUTOFF = 5, BASIS_VERB_DIST_CUTOFF = 5`
- Verb-noun distance cut-off for association: `VERB_NOUN_DIST_CUTOFF = 4`.

Table E1. The overlap of representative encoding of test pattern 'hatter,says,queen' and a variety of other encodable patterns.

Test pattern	Bin. pattern	Overlap
hatter,go,queen	1111110011	0.974595
hatter,say,king	0111100011	0.974003
turtle,say,queen	1111100001	0.973813
hatter,would,queen	1111000011	0.973788
hatter,go,time	1110110011	0.959719
hatter,think,queen	1111010011	0.955626
turtle,say,time	1110100001	0.950021
turtle,would,queen	1111000001	0.947137
hatter,would,time	1110000011	0.945508
hatter,go,king	0111110011	0.944756
turtle,say,king	0111100001	0.943421
hatter,would,king	0111000011	0.943293
turtle,go,queen	1111110001	0.942680
hatter,say,hatter	0011100011	0.942616
hatter,think,time	1110010011	0.937847
turtle,go,time	1110110001	0.927700
turtle,think,queen	1111010001	0.923638
hatter,think,king	0111010011	0.923143
turtle,would,time	1110000001	0.918514
head,say,queen	1111100000	0.918384
hatter,go,hatter	0011110011	0.917483
turtle,say,hatter	0011100001	0.913042
turtle,would,king	0111000001	0.912276
turtle,go,king	0111110001	0.907574
turtle,think,time	1110010001	0.905863

(Continued)

Table E1. (Continued).

Test pattern	Bin. pattern	Overlap
hatter,go,thing	1100110011	0.904857
hatter,would,hatter	0011000011	0.904360
head,say,time	1110100000	0.897983
head,would,queen	1111000000	0.895655
hatter,think,hatter	0011010011	0.891647
turtle,think,king	0111010001	0.886336
turtle,say,thing	1100100001	0.884995
head,go,time	1110110000	0.884820
turtle,go,hatter	0011110001	0.881117
head,think,queen	1111010000	0.881114
head,say,king	0111100000	0.880701
hatter,would,thing	1100000011	0.874690
hatter,think,thing	1100010011	0.874513
turtle,would,hatter	0011000001	0.874446
hatter,say,turtle	0001100011	0.872754
head,would,time	1110000000	0.870438
turtle,go,thing	1100110001	0.867190
head,think,time	1110010000	0.864927
head,say,hatter	0011100000	0.860190
turtle,think,hatter	0011010001	0.856023
head,would,king	0111000000	0.854352
hatter,go,turtle	0001110011	0.851418
hatter,say,head	0000100011	0.849715
turtle,would,thing	1100000001	0.841915
hatter,go,head	0000110011	0.841185
head,go,hatter	0011110000	0.840234
head,think,king	0111010000	0.838609
turtle,say,turtle	0001100001	0.838448
turtle,think,thing	1100010001	0.836987
head,say,thing	1100100000	0.831755
head,would,hatter	0011000000	0.827727
turtle,say,alice	1000100001	0.824155
head,go,thing	1100110000	0.823098
hatter,would,turtle	0001000011	0.821690
head,think,hatter	0011010000	0.818908
hatter,think,turtle	0001010011	0.815651
turtle,say,head	0000100001	0.815175
turtle,go,alice	1000110001	0.814121
turtle,go,turtle	0001110001	0.810484
turtle,go,head	0000110001	0.801212
hatter,think,head	0000010011	0.796473
head,think,thing	1100010000	0.795634
head,would,thing	1100000000	0.794096
head,say,turtle	0001100000	0.790379
head,say,alice	1000100000	0.789465
hatter,would,head	0000000011	0.789114
turtle,would,turtle	0001000001	0.786962
head,say,head	0000100000	0.777728
turtle,think,turtle	0001010001	0.775566
head,go,turtle	0001110000	0.773961
head,go,head	0000110000	0.772277
turtle,think,alice	1000010001	0.770818
turtle,would,alice	1000000001	0.766441
turtle,think,head	0000010001	0.757195
turtle,would,head	0000000001	0.754111
head,think,alice	1000010000	0.748169
head,would,turtle	0001000000	0.746573
head,think,turtle	0001010000	0.743799
head,would,alice	1000000000	0.739083
head,think,head	0000010000	0.733556
head,would,head	0000000000	0.725028

ORCID iDs

Lee J O'Riordan https://orcid.org/0000-0002-6758-9433
Myles Doyle https://orcid.org/0000-0001-7889-8357

References

[1] Cambria E and White B 2014 Jumping nlp curves: a review of natural language processing research *IEEE Comput. Intell. Mag.* **9** 48–57
[2] Harris Z S 1954 Distributional structure *WORD* **10** 146–62
[3] Zadrozny W 1992 On compositional semantics *COLING 1992 Vol 1: The 15th Int. Conf. on Computational Linguistics*
[4] Coecke B, Sadrzadeh M and Clark S 2010 Mathematical foundations for a compositional distributional model of meaning *Linguistic Analysis* **36** 345–84 (arXiv:1003.4394) *[cs, math]* (www.linguisticanalysis.com/volume-36/)
[5] Zeng W and Coecke B 2016 Quantum algorithms for compositional natural language processing *Proc. of SLPCS* **221** 67–75
[6] Coecke B 2019 The mathematics of text structure (arXiv: 1904.03478)
[7] Wiebe N, Kapoor A and Svore K 2015 Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning *Quantum Inf. Comput.* **15** 316–56 (www.rintonpress.com/xxqic15/qic-15-34/0316-0356.pdf)
[8] Giovannetti V, Lloyd S and Maccone L 2008 Quantum random access memory *Phys. Rev. Lett.* **100** 160501
[9] Arunachalam S, Gheorghiu V, Jochym-O'Connor T, Mosca M and Srinivasan P V 2015 On the robustness of bucket brigade quantum RAM *New J. Phys.* **17** 123010
[10] Di Matteo O, Gheorghiu V and Mosca M 2020 Fault-tolerant resource estimation of quantum random-access memories *IEEE Trans. Quantum Eng.* **1** 1–13 (https://ieeexplore.ieee.org/document/8962352)
[11] O'Riordan L J, Doyle M, Baruffa F and Kannan V 2020 QNLP: ICHEC quantum NLP toolkit (https:github.com/ICHEC/QNLP 10.5281/zenodo.3743034)
[12] Guerreschi G G, Hogaboam J, Baruffa F and Sawaya N 2020 Intel quantum simulator: a cloud-ready high-performance simulator of quantum circuits *Quantum Science and Technology* **5** 034007
[13] Nisbet R, Elder J and Miner G 2009 Part ii - the algorithms in data mining and text mining, the organization of the three most common data mining tools and selected specialized areas using data mining , *Handbook of Statistical Analysis and Data Mining Applications* eds R Nisbet, J Elder and G Miner (Boston: Academic Press) pp 119– 20
[14] Mikolov T, Chen K, Corrado G and Dean J 2013 Efficient estimation of word representations in vector space *1st International Conference on Learning Representations, ICLR, Workshop Track Proceedings May 2–4, 2013 Scottsdale, Arizona, USA* (arXiv: 1301.3781)
[15] Bird S, Loper E and Klein E 2009 *Natural Language Processing With Python* (Sebastopol, CA: O'Reilly Media Inc.) (www.oreilly.com/library/view/natural-language-processing/9780596803346/)
[16] Honnibal M and Montani I *et al* 2017 spacy 2: Industrial-strength natural language processing (NLP) with Python and Cython
[17] Blacoe W 2015 Semantic composition inspired by quantum measurement *Quantum Interaction*, eds H Atmanspacher, C Bergomi, T Filk and K Kitto (Cham: Springer Int. Publishing) *QI 2014 June 30 - July 3, 2014. Filzbach, Switzerland* (https://doi.org/10.1007/978-3-319-15931-7_4)
[18] Aerts D, Broekaert J, Sozzo S and Veloz T 2013 Meaning-focused and quantum-inspired information retrieval *QI 2013 July 25–27, 2013* Leicester, UK, Springer Berlin Heidelberg **8369** pp 71–83
[19] Wang B 2019 Dynamic content monitoring and exploration using vector spaces *Proc. 42nd Int. ACM Conf. on Research and Development in Information Retrieval SIGIR'19, New York, NY, USA, ACM*, 1444
[20] Jaiswal A K, Holdack G, Frommholz I and Liu H 2018 Quantum-like generalization of complex word embedding: a lightweight approach for textual classification *Proc. Conf. Lernen, Wissen, Daten, Analysen August 22–24, 2018 Mannheim, Germany* pp 159–168
[21] Tiwari P and Melucci M 2018 Multi-class classification model inspired by quantum detection theory (arXiv: 1810.04491)
[22] Wang B, Buccio E Di and Melucci M 2019 Representing Words in Vector Space and Beyond Representing words in vector space and beyond *Quantum-Like Models for Information Retrieval and Decision-Making* (Cham: Springer Int. Publishing) (https://doi.org/10.1007/978-3-030-25913-6_5)
[23] Wiebe N, Bocharov A, Smolensky P, Troyer M and Svore K M 2019 (arXiv:1902.05162) [quant-ph] Quantum Language Processing (arXiv: 1902.05162)
[24] Bausch J, Subramanian S and Piddock S 2019 A quantum search decoder for natural language processing arXiv e-prints, (arXiv: 1909.05023)
[25] Lambek J 2008 Pregroup grammars and Chomsky's earliest examples *J. Logic Lang. Inf.* **17** 141–60
[26] Smelyanskiy M, Sawaya N P D and Aspuru-Guzik A 2016 qhipster: The quantum high performance software testing environment (arXiv: 1601.07195)
[27] Schuld M 2017 Quantum machine learning for supervised pattern recognition *PhD Thesis* (https://hdl.handle.net/10413/15748)
[28] Mitarai K, Kitagawa M and Fujii K 2019 Quantum analog-digital conversion *Phys. Rev.* A **99** 012301
[29] Levy O and Goldberg Y 2014 Linguistic regularities in sparse and explicit word representations *Proc. 18th Conf. on Computational Natural Language Learning* (Ann Arbor, Michigan Association for Computational Linguistics), pp 171–80
[30] Socher R, Bauer J, Manning C D and Ng A Y 2013 Parsing with compositional vector grammars *Proc. 51st Annual Meeting of the Association for Computational Linguistics (vol 1: Long Papers)* pp 455–65 (www.aclweb.org/anthology/P13-1045)
[31] Mikolov T, Sutskever I, Chen K, Corrado G and Dean J 2013 Distributed representations of words and phrases and their compositionality *Proc. 26th Int. Conf. on Neural Information Processing Systems—vol 2*, NIPS'13, *Dec 5 –10, 2013 Lake Tahoe, NV* Curran Associates Inc. pp 3111–3119
[32] Hagberg A A, Schult D A and Swart P J 2008 Exploring network structure, dynamics and function using networkx *Proc. 7th Python in Conf. (SciPy2008) Aug 19–24, 2008* Caltech, Pasadena, CA, USA, eds G Varoquaux, T Vaught and J Millman pp 11–16 (https://conference.scipy.org/proceedings/scipy2008/paper_2)
[33] Trugenberger C A 2001 Probabilistic quantum memories *Phys. Rev. Lett.* **87** 067901
[34] Trugenberger C A 2002 Quantum pattern recognition *Quantum Inf. Process.* **1** 471–93
[35] Barenco A, Bennett C H, Cleve R, DiVincenzo D P, Margolus N, Shor P, Sleator T, Smolin J A and Weinfurter H 1995 Elementary gates for quantum computation *Phys. Rev.* A **52** 3457–67
[36] Harrow A W 2020 Small quantum computers and large classical data sets (arXiv:2004.00026)

[37] Coecke B and Duncan R 2011 Interacting quantum observables: categorical algebra and diagrammatics *New J. Phys.* **13** 043016
[38] Cowtan A, Dilkes S, Duncan R, Krajenbrink A, Simmons W and Sivarajah S 2019 On the qubit routing problem *14th Conf. on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019), vol 135 of Leibniz Int. Proc. in Informatics (LIPIcs) Jun 3 – 5, 2019 Maryland, USA*, eds W van Dam and L Mancinska, pp 5:1–32
[39] Coecke B, de Felice G, Meichanetzidis K and Toumi A 2020 Quantum natural language processing (16 April 2020) (https://medium.com/cambridge-quantum-computing/quantum-natural-language-processing-748d6f27b31d)
[40] de Felice G, Toumi A and Coecke B 2020 DisCoPy: Monoidal categories in Python p 2020 (arXiv: 2005.02975)
[41] Jakob W, Rhinelander J and Moldovan D 2017 (pybind11—seamless operability between C++11 and Python) (https://github.com/pybind/pybind11)
[42] Hunter J D 2007 Matplotlib: A 2d graphics environment *Comput. Sci. Eng.* **9** 90–5
[43] van der Walt S, Chris Colbert S and Varoquaux G 2011 The numpy array: a structure for efficient numerical computation *Comput. Sci. Eng.* **13** 22–30
[44] Virtanen P *et al* 2020 SciPy 1.0: fundamental algorithms for scientific computing in Python *Nat. Methods* **17** 261–72
[45] McKinney Wes 2010 Data structures for statistical computing in Python *9th Python in Science Conference June 28 – July 3, 2010 Austin, Texas* (https://doi.org/10.25080/Majora-92bf1922-012)